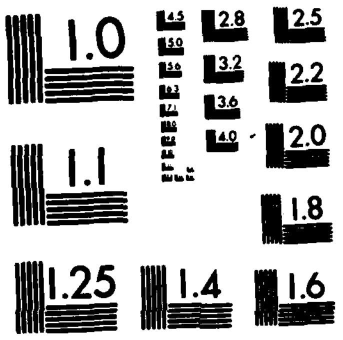
AD-A138 174   DECENTRALIZED CONTROL OF SCHEDULING IN DISTRIBUTED SYSTEMS(U) MASSACHUSETTS UNIV AMHERST DEPT OF ELECTRICAL AND COMPUTER ENGINEERING J A STANKOVIC   1/1

UNCLASSIFIED   15 DEC 83 DAAB07-82-K-J015   F/G 9/2   NL

END
FILMED
3-84
DTIC


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(2)

RESEARCH AND DEVELOPMENT TECHNICAL REPORT
CECOM

DECENTRALIZED CONTROL OF SCHEDULING IN DISTRIBUTED SYSTEMS

John A. Stankovic
Dept. of Eletrical & Computer Engineering
University of Massachusetts
Amherst, MA 01003

DAAB07-82-K-J015

Annual Report for the Period 16 DEC 82 to 15 DEC 83

Distribution Statement

Approved for public release;
distribution unlimited

Prepared for
Center for Communications Systems (C. Graff)

CECOM
U S ARMY COMMUNICATIONS-ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703

Table of Contents

1.0 Introduction

This report summarizes the work and results produced during this contract over the past year. A summary of our results on the use of Bayesian Decision theory ~~are~~ is reported in Section 2.1. Two master's thesis were completed: one on bidding (Section 2.2) and one on stochastic learning automata (Section 2.3). We have also been working on an analytical model ~~(Section 2.4)~~ but little new progress over that reported in the last quarterly report has been made in this area. Progress was made on a decentralized scheduling approach for systems with tasks having real-time constraints, and this is summarized in Section 2.5.

This report also briefly states our overall conclusions (Section 3) and makes several recommendations (Section 4) for the work to be done in the final year of this contract. Section 5 lists the papers, thesis~~es~~ and presentations produced this year. Section 6 contains the required budget information, and finally the report ends with a copy of the distribution list.

2.0 Main Text

2.1 Bayesian Decision Theory

The form of decentralized control under investigation in this grant has not been dealt with to any large degree because of its intractable nature, although it is becoming increasingly important for distributed computer systems. Bayesian Decision theory is a (relatively) low cost heuristic for decentralized control of job scheduling when no a priori information is known about jobs. Using extensive simulations, our heuristic is shown to provide good response time and load balancing in comparison to analytical and baseline simulation models. The amount of movement necessary to achieve this performance is quantified. A major reason for our heuristic performing well is that it is able to adapt to the quality of the state information it is receiving and use that in making its decisions. This ability becomes increasingly important for busy or noisy subnets and we hypothesize also for larger and larger networks.

Several specific observations of our simulations include: that once tuned the algorithm works in a stable manner over a wide variety of conditions and over a large number of runs, that the probability distributions describing the true states of nature and the likelihoods of an observation converge to values representing the level of observability of the system, that the loss of monitor nodes does not cause major problems, that the heuristic works well under light loads but simpler approaches would be just as good, that the need for the heuristic increased both for moderate loads and for a decrease in the accuracy of state information available, and several thresholds included in the heuristic improve the operation of the algorithm.

As a final suggestion we believe that the heuristic should be modified to avoid moving very large jobs. For example, when our heuristic decides to move a job, it removes a job off the back of the local queue without regard for size. A better approach would be to first check if this job is very large, if so, move some other job. This should further improve response time.

2.2 Bidding

Decentralized process scheduling (as opposed to job scheduling) using many factors is a complicated issue. Which factors have significant impact on performance (response time), which do not, how do various factors interrelate, and how are the three questions just posed answered under various system conditions are important but difficult questions. In this research we have developed a decentralized scheduling algorithm based on bidding in which it is easy to vary the factors and their interrelationships. A simulation model has been implemented to test the variations of the algorithm under any number of system conditions. Simulation results can be compared to simple analytical models and baseline simulations, as we did. For example, one baseline used was where 100% accurate data is used instead of delayed information. Our current simulation results indicate that (under the conditions tested), (1) bidding is quite effective even with simple parameters, (2) the use of cluster and distributed group information in bidding is only marginally worthwhile (but its importance is expected to increase as the percentage of processes exhibiting cluster and group characteristics increase), and (3) transmitting bids a distance which is a function of the system load is effective. Of course, with such a complicated simulation model the results have to be considered preliminary until a much wider set of conditions and combination of parameters are tested.

More detailed results are reported in a Master's thesis entitled, "Decentralized Process Scheduling in a Distributed Computing Environment". Work is continuing in this area and the simulation program is being modified to handle more interesting situations and to automatically display the results while the simulation is in progress.

2.3 Stochastic Learning Automata

A job scheduling algorithm based on a Stochastic Learning Automata (SLA) model has been developed. We have made extensive additions to this model in order to better satisfy our requirements. The most important of these additions is an external pattern recognition capability by virtue of which the SLA is able to determine good actions for each of the network states it can recognize (e.g., what hosts are overloaded on underloaded).

To achieve confidence in the learning scheme we have proposed, extensive simulations have been carried out. Relative performance of various schemes have been analyzed and results are compared to simple baseline and analytical models.

Essentially, a network of five hosts was modeled using GPSS, an event driven simulation system because of the ease of modeling arrivals, departures and gathering of queue statistics.

The scheduling decisions are performed periodically by the SLA routines and also when a process finishes. Fortran was used to implement the algorithmic part because of the cumbersome nature of implementing complicated control structures in GPSS.

State information is allowed to percolate through the network. By this we mean that each host periodically sends it's state to it's neighbors. If a host is two hops away from the sender, it will receive this information only after two state update periods.

As we have added the capability of state recognition in the SLA model, an important question that arises is: how many states are appropriate for our problem? We have tested our system under three different "number of states". They are:

1. 10 states: In this system, a host, e.g., Host A can recognize states like 'BC underloaded', BD underloaded', etc......, 'DE underloaded', 'B underloaded',...., 'E underloaded', where A,B,C,D,E, are the hosts.

2. 4 states: In this system, host A can recognize states like 'B underloaded',.....,'E underloaded'. A similar situation exists in all the other hosts.

3. 1 state: This system corresponds to the case of the SLA without any pattern recognition capability, i.e., a host does not have information about the states of other hosts.

The metrics of performance that we were interested in monitoring are as follows:

1. Average loading of the hosts in the network and response times.

2. The number of jobs that have to be moved to achieve the balancing.

3. The reaction time of the network to sudden surges of jobs at different points in the network.

4. The performance of the automata in the absence of the associative memory.

5. To test various reinforcement schemes such as Reward-Penalty and Reward-Inaction.

6. To determine the effect of global versus pairwise response feedback from the environment.

Under the assumptions that we have made in our studies, we have derived results that have indicated the following facts to us:

1. Load balancing in networks does improve performance in terms of average response times as compared to a system that does not try to balance the network. It also ensures reasonable utilization of resources overall as a consequence.

2. Given the high degree of uncertainty prevalent in such networks, conventional, static load balancing schemes (which might utilize the statistical behavior of the network as an example) do not seem very appropriate. This was illustrated by the performance of the fractional assignment model which was described in the thesis,

"Decentralized Scheduling Using a Network of Stochastic Learning Automata." This model utilized static distribution of jobs to generate assignments and did not perform as well as our SLA model.

3. The 10 state system performed the best of the SLA models that we studied under a majority of conditions. However, a key shortcoming of this system is that it performs well only when the scheduling period is 2000 ms.

4. The 4 state system, on the other hand, does not perform quite as well as the 10 state system, but is less sensitive to changes in scheduling period. It performs best when scheduling is performed at 4000 ms. Although we have factored in scheduling overhead in our model, it may be possible that in real systems of interest to us, scheduling overhead is higher than we assume in our simulation model. In such a case, it might be more appropriate to use the 4 state system.

5. The 1 state system which is the SLA in it's primitive form, performed quite poorly as compared with the 4 and 10 state systems. This shows that significant degradation in performance results because of a lack of an external pattern recognition capability. The tradeoff involved here was apparent: To recognize states, the hosts must exchange state information. This would tend to reduce the useful bandwidth of the subnet. However, we have conclusively shown that it is a good idea to have this state information transfer.

6. Our tests on the variation of learning constants which are integral parts of the SLAs indicated that the 4 and 10 state systems were not very sensitive to these for the range over which we varied these.

7. Under conditions of surges, we have seen the 4 and 10 state systems perform quite well, with the 10 state system having a more consistent performance. This indicates the stability of the algorithm in the face of random events. As a matter of fact, we have not seen any other results in scheduling literature to date that have incorporated this feature.

8. We have seen that the performance of the system degrades as the state update period increases. This indicates that although information does not have to be accurate, it cannot be so stale that a unique association cannot be derived between state and action over a finite number of trials. One consequence of this could be that hosts may not be more than 3 or 4 hops apart.

9. The effect of perfect knowledge on an algorithm called Omni was determined. It was seen that under the few tests we performed on it, Omni either performed as well as the best SLA model or did worse. This emphasized the need for a smart decision making algorithm to be coupled with this perfect information.

10. From our tests that we conducted using an implicit global reward scheme we learned that in most cases the pairwise computation of

reward performed better than the global scheme. However, the concept of an implicit global feedback might still be appropriate under different conditions.

The following are some of the essential characteristics of a good decentralized control algorithm. We feel that our algorithm has been able to satisfy these quite adequately:

1. The algorithms implemented by multiple, cooperating, physically distributed SLA's,

2. there are no periods of centralized control nor is there any centralized state information.

3. the algorithm dynamically adapts to the network states and learns to associate an optimal action with each state,

4. the algorithm performs efficiently with noisy and delayed information. However, we have seen that state information cannot be delayed for an arbitrarily long time. This limits the applicability of this scheme to small and medium size networks unless some kind of partitioning scheme is resorted to,

5. we have seen the algorithm recover gracefully from random events like the various kinds of surges that we introduced,

6. there is no bias towards any one or more hosts and on the average, the network is reasonably well balanced,

7. there is very low overhead in running this algorithm.

Thus we have seen that it is possible to design a controller that can acquire the characteristics of the system for jobs scheduling "on line" while starting with no a priori information about the dynamic behavior of the network. The controller is able to associate an optimal action with every state it can recognize after a reasonably short period of time.

We have also seen that is is possible for a network of controllers to cooperate with each other to improve the overall performance of the network. Under most conditions, the algorithm performed reasonably good balancing and also generated quite respectable response times, as we later verified by comparisons with some analytical models.

The algorithm is also able to recover gradually from sudden surges of jobs. We feel that this has been a very strong plus for our system. The algorithm performs as it does in the face of old and noisy state information and uncertain feedback from the environment.

In comparison with some of the analytical models we studied, we found that the SLA based system performed quite adequately, especially since most of these analytical models make some very unrealistic demands in terms of a priori information that is needed by them.

2.4 Analytical Model

Significant effort has gone into developing an analytical model during this past year. The states of the model was reported in the last quarter's report. No significant progress in this area was made in the last three months. However, we have investigated the use of Kalman filters as part of the analytical model. While this approach seems promising no significant results can be reported yet.

2.5 Real Time Constraints

A simulation model for a decentralized scheduling algorithm based on bidding, but where tasks have real-time constraints, was designed and implemented. This simulation program is currently being debugged. The previous quarter's report describes the algorithm itself and some preliminary simulation results for our scheduling algorithm on one host. Results on the completely decentralized model should be forthcoming.

3.0 Conclusions

The major conclusions to date include showing that Bayesian decision theory, stochastic learning automata and bidding are all valuable approaches to scheduling in distributed systems under very demanding conditions. Our results also identify various conditions where one approach is more suitable than the other. In the process a number of insights have been attained including:

1. Varying the distance a bid travels as a function of load improves performance,

2. as delays in the subnet increase it becomes more and more important to use a technique like Bayesian Decision theory which can explicitly handle this information,

3. it is effective to try to spread work among all the lightly loaded hosts but this needs to be done without causing too much job movement,

4. the overhead of the scheduling algorithm itself was a first order effect on performance hence it must be kept low, and

5. a network of stochastic learning automata (SLA) quickly converge to the best actions to take for a given topology and subnet delay and give good performance assuming delays are not too bad. This later observation seems to limit the use of SLAs to small networks or to small clusters of nodes in a larger network.

4.0 Recommendations

Our next step is to continue our efforts on fully developing the process scheduling technique based on bidding. This approach can incorporate many more interesting factors in the scheduling algorithm. We also intend to more specifically compare the various approaches under the same conditions. Finally, continued effort will be made in developing and validating (via simulation) an analytical model.

5.0 Papers Thesis', and Presentations Produced this Year

5.1 Papers

1. "An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups," with Inderjit Sidhu, submitted to the 4th International Conference on Distributed Computing Systems, September 1983.

2. Dynamic Task Scheduling in Distributed Hard Real-Time Systems," with Krithi Ramamritham, submitted to the 4th International Conference on Distributed Computing Systems, August 1983.

3. "Bayesian Decision Theory and Its Application to Decentralized Control of Job Scheduling," submitted to IEEE Transactions on Computers, May 1983.

4. "Current Research and Critical Issues in Distributed Software Systems," with K. Ramamritham, and W. Kohler, submitted to ACM Computing Surveys, March 1983.

5. "Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms," submitted to Computer Networks, January 1983.

6. "A Heuristic for Cooperation Among Decentralized Controllers," Proceedings INFOCOM 83, invited paper, April 1983

5.2 Thesis

1. Sidhu, I., "Decentralized Process Scheduling in a Decentralized Computing Environment," Master's Thesis, University of Massachusetts, July 1983.

2. Mirchandaney, R., "Decentralized Job Scheduling using a Network of Stochastic Learning Automata," Master's Thesis, University of Massachusetts, December 1983.

5.3 Presentations

1. IEEE Real Time Systems Symposium, Panel Member, Topic - Critical Issues in Real Time Operating Systems, December 1983.

2. Presented Tutorial at Symposium on Reliability in Distributed Systems, Clearwater Beach Florida, October 1983.

3. IEEE First Workshop on Real Time Operating Systems, Paper Presentation, Niagara Falls, NY, August 12, 1983.

4. INFOCOM83, Paper Presentation, San Diego, California, April 19, 1983.

6.0 Budget for 2nd Year of Contract

6.1 Current Budget

	Planned	Actually Spent
December (82)	3,857.06	3,857.06
January (83)	2,575.83	2,474.83
February	3,412.64	3,412.64
March	449.30	449.30
April	1,835.66	1,834.66
May	1,142.33	1,142.33
June	4,592.00	4,592.00
July	5,779.50	5,779.50
August	7,000.00	2,517.11
September	7,558.71	1,114.99
October	5,876.17	3,041.70
November	5,876.17	5,911.03
December	5,876.16	
	55,830.53	36,228.15

6.2 Cummulative Cost to Date

1. 30,236.47 (first year expenditure) + 36,228.15 = $66,464.62

6.3 Revised Budget for Third Year of Contract

Spent to Date - $66,464.62

December (83)	4,000
January (84)	4,000
February	4,000
March	4,000
April	4,000

May	4,000
June	7,000
July	7,000
August	7,000
September	5,000
October	5,000
November	5,000
December	4,365.38
	$130,830.00

7.0 Appendix

This appendix contains two papers produced in the last quarter. Both have been accepted for publication in the 4th International Conference on Distributed Computing Systems.

1 ~~6~~ 11

An Adaptive Bidding Algorithm For Processes, Clusters and Distributed Groups

John A. Stankovic
Inderjit S. Sidhu

Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, Mass. 01003

ABSTRACT

Bidding algorithms have been accepted as a possible heuristic for cooperation among distributed components. This paper describes an adaptive bidding algorithm for decentralized process scheduling in computer networks. The algorithm has the potential to use process resource requirements, special resource needs, process priority, precedence constraints, the need for clustering and distributed groups, specific features of heterogeneous hosts, and various other process and network characteristics in performing process scheduling. The algorithm also dynamically adjusts the distance to which a request for bid is transmitted. A simulation program to evaluate our algorithm has been implemented and the preliminary results are presented. The results indicate that good performance is possible with our technique. An ultimate goal is to determine the relative effectiveness of the multitude of various system parameters that could be used in bidding algorithms.

1.0 INTRODUCTION

Bidding algorithms have been accepted as a possible heuristic for cooperation among distributed components. Scheduling in highly cooperative distributed systems [9] is an application area ideally suited to bidding although, to date, it has not been effectively evaluated. In this paper we propose a sophisticated and adaptive bidding algorithm to deal with scheduling in highly cooperative computer networks. The algorithm is sophisticated because it attempts to match processes to processors based on many factors including process resource requirements, special resource needs, process priority, precedence constraints, the need for clustering and distributed groups (opposite of a cluster), specific features of heterogeneous hosts, and various other process and network characteristics to be described below. We use the framework of a McCulloch Pitts neuron to describe this portion of the algorithm. The algorithm is adaptive in the sense that bids are transmitted a distance (measured in hops) that is a function of the load on a host, its collection of neighboring hosts, and possibly previous responses to bids.

The evaluation of our algorithm is done by simulation of which some preliminary results are presented here. The preliminary results indicate that bidding is effective using even simple parameters, that adapting the distance a bid travels is a useful feature, and that the use of cluster and distributed group information improves system operation. More simulations are needed to determine which factors have a first order effect on performance.

The advantages of our approach are that it is easily extensible to include any parameter deemed important for decision making, that it attempts to load balance in nearest hosts first, it includes resource, clustering and distribution requirements, and that tuning for a given network and application load is possible. The main disadvantage is that it may be difficult to choose the proper set of parameters for the scheduling algorithm and their values for a given system. However, today's uniprocessor operating systems typically have hundreds of parameters whose values are tuned for a given installation. While there is no science for choosing the values, experience has enabled effective choice of these parameters. We feel the same will be true of the parameters in an adaptive load balancing algorithm such as the one proposed here.

The remainder of the paper is organized as follows. Section 2 presents a brief summary of related work in scheduling on distributed systems. Section 3 describes our view of a process and process scheduling. Section 4 describes our bidding algorithm. For this short paper only a brief description of our simulation model can be given here. This is found in section 5. See [17] for a complete description. The simulation results are presented in section 6 and tentative conclusions are drawn in section 7.

2.0 BACKGROUND

Much of the research on scheduling for distributed systems can be considered "task assignment" research and can be loosely classified as either graph theoretic [2][23][24], queueing theoretic [1][6], based on mathematical programming [7][13][15], or heuristic [4][5][8][10]. In most

2 of 11

of these cases a task is considered composed of multiple modules and the goal is to find an optimal (or in some cases suboptimal) assignment policy for the modules of an individual task. Typical assumptions found in task assignment work are: processing tasks are known for each module of the task, the interprocess communication costs (IPC) between every pair of modules is known, IPC is considered negligible for modules on the same host, and reassignment is not possible. Many times these assumptions are not practical.

Other scheduling research is based on job scheduling where no a priori knowledge about jobs is known. Here, job scheduling refers to the fact that jobs have not yet begun execution when the scheduling decisions are made. Approaches based on queueing theory [1], statistical decision theory [20][22], or other heuristics [21] have appeared. This paper treats the more difficult problem of process scheduling.

There has been limited scheduling research based on bidding [11][12] where specific tasks are matched to processors based on the current (or predicted) ability of the processors to perform this work. In [12] bids were made based only on percentage of free memory at a site. A more sophisticated study of bidding [18] has appeared, but it was not evaluated and was not applied to scheduling in distributed systems. However, more and more effort is being made to use bidding because of its potential usefulness in highly cooperative and decentralized systems.

Other research in scheduling on networks has been performed in the context of operating systems. For example, published work includes StarOS [14], Medusa [16], Roscoe (now called Arachne) [19], the domain structure operating system [3], and MICROS [25]. These systems tend to have heuristics as scheduling algorithms, but evaluations of these scheduling algorithms are sketchy. Further, only StarOs and Medusa treat process clustering issues. We intend to stress the evaluation and also include clustering and other issues.

3.0 PROCESS STRUCTURE

Our conception of a process is derived from that of Casey and Shelness [3]. A process is an execution sequence through domains, where a domain is the process' collection of segments (objects) at any point in time. A segment is defined as a logical unit of information with identical access characteristics. There are code, data, and environment segments. Depending on their characteristics, segments can be shared simultaneously, unshared or shared sequentially between domains and processes. Casey and Shelness restricted their work to intra-process concerns. We extend their concept of a process to include inter-process communication. That is, various forms of send and receive primitives are supported, with the system determining the location of processes. Depending on the frequency and amount of interaction between sets of processes, clusters and distributed groups might arise.

A cluster is defined to be a set of processes that communicate frequently or in large volume, or both, in such a way that it is beneficial to schedule them on the same host. A cluster is static if the collection of processes comprising the cluster is independent of time, otherwise it is dynamic. Individual processes may belong to any number of clusters. A collection of clusters which interact with each other is known as a cluster group. Clusters forming cluster groups do not interact with each other to any large extent because, if they did, they would be considered a single cluster. Our bidding algorithm attempts to perform dynamic site assignment for static and dynamic clusters. Cluster groups are not treated in any special way but are subject to the below described bidding algorithm. If an entire cluster (group) is assigned to the same host it is referred to as a placed-cluster (placed-group). Such an assignment does not occur automatically upon identifying a cluster (group) but is subject to the current state of the system as described in the section 4.

Actual identification of what tasks comprise a cluster and the potential benefits if this cluster becomes a placed-cluster is difficult. Our algorithm assumes that there is some monitoring process that is separate from the scheduling algorithm and that can perform this function. This monitoring process then provides the necessary input to the scheduler (statically or dynamically). Users have the capability of identifying clusters and groups directly to the scheduler.

A distributed group is a collection of processes that communicate with each other but would benefit from the inherent parallelism of being placed on separate hosts (all other things being equal). A distributed group is the opposite of a cluster. Totally disjoint processes might also benefit by being placed on separate hosts but they do not communicate with each other and therefore are not considered a distributed group. Similar to clusters, users or a monitoring process informs the scheduler which processes constitute a distributed group. Proportional costs and benefits are necessary for different combinations of separation. The bidding algorithm handles the use of such information.

A process scheduling algorithm has the responsibility for choosing the host on which to execute a process, cluster or distributed group at a given point in time. A process may move any number of times during its execution up to some predefined maximum. The bidding algorithm presented here is such a process scheduling algorithm.

4.0 BIDDING ALGORITHM

The bidding algorithm is replicated at each host of the network and is invoked periodically. The major functions of the algorithm are:

1. deciding on whether to transmit requests for bids,
2. transmitting bid requests,
3. processing requests for bids from other sites,
4. transmitting tasks, and
5. dispatching local tasks.

The algorithm makes use of a modified McCulloch Pitts neuron calculation both to determine feasible candidates for movement, as well as to respond to request for bids. We describe this first, before describing each of the above steps.

McCulloch Pitts Evaluation Procedure

The McCulloch Pitts evaluation procedure runs as a subroutine. The McCulloch Pitts neuron (modified slightly) (Figure 1) is a decision cell with excitations and inhibitors as inputs, and one output (a value which is the sum of all the excitations or zero). If any of the inhibitors is set, the output is zero. In our context, inhibitors are: this process has moved y times already, this process requires a local resource that is not available elsewhere, the user fixes the process on this host, etc. Excitations are any parameters that are important in the scheduling decision - potentially included are all process and network characteristics (as described below). A goal of this study is to determine the usefulness of various individual parameters and in combination. Of course, when multiple parameters are used the various excitations they cause have to be normalized. The excitations used for the simulations presented in this paper are given in the descriptions of algorithms 1-6.

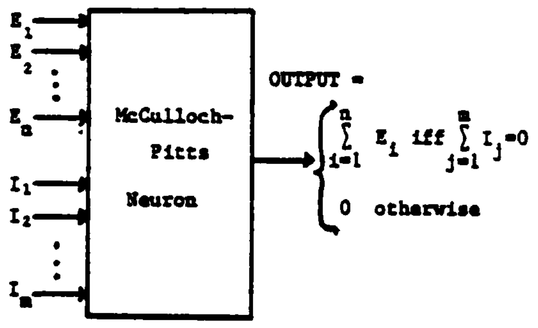

E_1
E_2
E_n
I_1
I_2
I_m
McCulloch-Pitts Neuron
OUTPUT =
$\begin{cases} \sum_{i=1}^{n} E_i & \text{iff } \sum_{j=1}^{m} I_j = 0 \\ 0 & \text{otherwise} \end{cases}$

Figure 1: Modified McCulloch Pitts Neuron

1. Deciding on whether to transmit requests for bids

Local processes of a host are evaluated via a modified McCulloch Pitts neuron calculation to determine how well it is executing on the current host. All processes above a threshold are well suited to the current host and are not candidates for movement. Those processes evaluated below a threshold are performing poorly and are potential candidates for movement, implying that requests for bids might be transmitted for them, subject to the constraints described below. State information used in the evaluation includes both process and network characteristics.

Process characteristics

Process characteristics are essentially those features that characterize the state of an active process. They include:

1. Process state (ready, running or blocked)
2. Cluster and distributed group information.
3. Process size (memory requirement). This is a sum total of the size of current segments a process is using. This includes code, data and environment segments and is biased by the state of the resource segments and how it is being used (e.g., if the state is 'shared simultaneously' then there is a bias for just how many domains are currently using it).
4. Total cpu-time requirement.
5. Time used so far.
6. Number of times moved.
7. Priority.
8. Position in wait queue for resources.
9. Precedence Relationships (if any).
10. Locks held by the process.
11. The need for special resources.

Some of the above mentioned process characteristics are static and some are dynamic. We assume that the required information is either provided by the user (e.g., through some job control language) or gathered by the operating system dynamically. Each characteristic used is assigned an excitation (may be adaptive) or inhibitor value. Further, to make the system work even with missing information, default values are defined for the various characteristics where needed, or just treated as null.

Network Characteristics

These are a host's perception of the state of the network including itself. Like process characteristics some of these are static and some are dynamic. The network characteristics maintained by a host i include:

1. Total and available primary memory at the site.

2. Processor speed of the site.

3. Cost of using a link between hosts i and j.

4. List of neighbors of host i. This is fixed, except when some site crashes.

5. List of fixed resource segments at the site and their characteristics (e.g., unshared, shared sequentially, or shared simultaneously).

6. Number of currently active processes at the site.

7. Current busyness of each site based on some utilization factor.

8. Number of jobs moved to/from this site within the last t.

9. Number of incomplete clusters, and distributed groups at the site and their IDs.

10. The maximum number of processes that this site can move at one invocation of the scheduling algorithm (for stability and constant worst case running time).

Again, each of the network factors actually used in a given algorithm is assigned an excitation or inhibitor value.

<u>2. Transmitting bid requests</u>

The process with the lowest value (but not zero), as outputted from the McCulloch Pitts neuron, is the most eligible process for movement. If the lowest value is above a threshold, then there are no eligible candidates. Depending on a system parameter, requests for bids may not be transmitted at all or are transmitted for:

1. the single most eligible process,

2. the most eligible processes up to a certain maximum, or

3. all eligible processes.

The distance (measured in hops) to which request for bids are transmitted is an adaptive parameter which lies anywhere from 0 (no requests sent) to the maximum diameter of the network depending on the current level of busyness of a host, its neighbors, and previous responses to bids for this process. Since this part of the algorithm is not stressed in the simulation results presented here and to conserve space, we do not described the adaption process in any detail.

<u>3. Processing Request for Bids</u>

Bid requests contain the characteristics of tasks so the host receiving a request for bid merely runs the McCulloch Pitts evaluation procedure and returns its output (assuming it is above the threshold implying that it would run well at this site) as the bid, else it does not bid. This is essentially matching a set of process characteristics to a hosts' view of the network. A variation that is being investigated is to have a host take into account outstanding bids when making subsequent bids. For example, a host might assume that a percentage of outstanding bids will be accepted and thereby adjust its new bid to account for this possibility.

<u>4. Transmitting Tasks</u>

A host waits for time delta t (currently a function of the distance the bid travels) for responses to requests for bids. A received bid is adjusted by incrementing it by a number proportional to the cost of moving the process to the designated site. After the adjustment, the best bid wins and the process is transmitted assuming that the bid is above a threshold and that the process is still performing poorly at its current site. If no bids are suitable and the process is still performing poorly, then the distance to which bids are sent is incemented by one and new request for bids are transmitted. This is one way in which the distance a bid propagates is adjusted.

<u>5. Dispatching</u>

The highest priority ready task is dispatched for one time slice. In the simulation, multiprogramming is supported with blocking possible due to lack of primary memory.

5.0 BRIEF DESCRIPTION OF SIMULATION

Our simulation program is written in GPSS and Fortran and can handle any number of hosts in any topology. Each host runs the decentralized process scheduling algorithm with bidding. The model is flexible enough to develop variations of the general bidding algorithm, such as by using different parameters in the McCulloch-Pitts evaluation process. Six such algorithms are presented in this paper. The first three are very simple in order to help tune the simulation model, to determine how well bidding works with very limited state information, and to illustrate the process of normalizing factors in the McCulloch Pitts neuron. The last 3 algorithms are sophisticated enough to deal with clusters and distributed groups - issues important in highly cooperative distributed systems.

1. <u>Algorithm 1</u> - In this algorithm the scheduling decision is based only on the size of the process. The algorithm tries to assign larger processes to sites that have have more free memory so that the blocking of processes due to nonavailbility of memory is reduced. The McCulloch-Pitts neuron uses

5 of 11

C1 = (Average process size/size of this process) * (1/ Normalized fractional memory utilization of a host)

as the bid estimate.

2. Algorithm 2 - In this algorithm, a matching of process and network characteristics is done by making an attempt to match longer processes (those that require more cpu time) to hosts that are relatively less busy, as determined by an estimate of their current queue length. The criterion used by the McCulloch-Pitts neuron is

C2 = (Process time requirement/ Queue length at host making the bid) * Normalization factor.

(A queue length of 0 is considered to be equivalent to a queue length of 0.25 to avoid dividing by zero). This algorithm tries to balance the load over the network.

3. Algorithm 3 - Algorithm 1 attempts to improve response time without any consideration for load balancing. Algorithm 2 tries to balance the load but without taking into account memory requirements. Both of these algorithms are too trivial. Algorithm 3 combines algorithm 1 and 2 with suitable weights, i.e., it takes both the size of the process and load balancing into account. Thus the criterion used by the McCulloch-Pitts neuron here is

$$C3 = (k * C1) + (1 - k) * C2.$$

The value of k is a system tunable parameter, which varies for the different arrival rates. It is tuned by making several runs.

Algorithm 1, 2 and 3 employ very basic scheduling criteria and are used to determine how well bidding works with limited state information. However, they illustrate the procedure for combining and normalizing various state information. For example, eventually we would like to combine most of the process and network characteristics into one formula that inherently adapts to a wide range of loads and conditions by taking into account all the important state information. This means that a given host would be using a network wide view in making scheduling decisions. Of course, if any state information is determined to have only a second order effect on performance it can be eliminated from the McCulloch Pitts Neuron calculation. Algorithms 4, 5 and 6 pertain to the more sophisticated criteria of distributing processes or clustering them in groups.

4. Algorithm 4 - Algorithm 4 is an extension of algorithm 3 where we simulate that a certain percentage of processes that enter the system are members of distributed groups. In the simulation distributed groups of size 3, 4 and 5 are generated with equal probability and successively generated sets of 3 processes each are sent to the various sites in the network with equal probability, so that the processes get evenly distributed and yet some members of distributed groups end up on the same site. Since members of a distributed group should preferably be on different hosts, but do not have to be, the algorithm tries to separate the members. It does this by reducing the bid estimate by a value proportional to the number of members of the same distributed group currently residing at that site. Thus, the McCulloch-Pitts neuron criterion is

$$C4 = C3 - (p * K)$$

where, p is the number of currently local members of the same distributed group and K is a system tunable parameter.

Note, however, that separation may not occur or may come about only partially for some distributed groups because of other considerations, such as criteria C1 and C2. That is perfectly acceptable, and members of distributed groups may execute on the same host. However, those members that do execute simultaneously on the same host are simulated as suffering a penalty by increasing their running time in proportion to the number of distributed group members currently at that particular host. Thus, if the total number of local members of a distributed group is p, the execution time of the member that is running is multiplied by p for the duration of its remaining run or the current time slice, whichever is shorter.

5. Algorithm 5 - Algorithm 5 is an extension of algorithm 3, where we add clusters to the system. Clusters of size 3, 4 and 5 are generated with equal probability and the members are sent to the different sites with equal probability. The algorithm tries to assemble the members of the cluster on the same site. This is done by combining a bid with an additional constraint. Whenever, a bid request is received for some process of a cluster group, the total size (memory requirement) of all its cluster mates at the site making the bid is evaluated. This memory requirement is transmitted along with the bid. The bid is calculated as part of the McCulloch Pitts calculation and it is:

$$C5 = C3 + q * (\text{ memory requirement })$$

where, q is a system tunable parameter and the memory requirement is the total memory required for all segments of all processes of the same cluster that currently reside at the local site. When bids are evaluated, the host originally sending the request for bids then sums the memory requirements of all the different pieces of the cluster (obtained from hosts responding and containing elements of the cluster).

The site with the best bid and containing enough free memory to hold all processes of the cluster known to the site requesting a bid is

6 of 11

chosen as the site for the formation of the cluster. This approach attempts to form the cluster at the site with the largest accumulated size of members of the same cluster (via the bid itself) and not the site with the maximum number of cluster members. It also tries to avoid instabilities by uniquely choosing the same site for formation of the cluster (bid plus enough free memory) even if multiple sites bid simultaneously.

As with distributed groups, processes of a cluster can still end up running on different hosts. In that case we simulate the penalty suffered by members of the cluster as an increase in running time proportional to the difference in the current size of cluster and the size of the complete cluster.

6. Algorithm 6 - Algorithm 6 is a combination of all the previous algorithms. It combines the considerations of memory requirements, queue lengths, distributed groups and clusters. Thus the criterion for the McCulloch-Pitts neuron here is:

$$C6 = C4 + C5$$

Note that the direct addition of C4 and C5 is possible because algorithms 4 and 5 affect mutually exclusive sets of processes, i.e., we have assumed that any process which is part of a distributed group is not also part of a cluster group, and vice-versa.

The operation of these algorithms is studied over various arrival rates (Table 1), subnet delays (from 25 msec per packet to 200 msec per packet), and scheduling intervals (from 2 to 16 secs). Also tested is the effect of sending bids a fixed distance of 1 hop, versus a fixed distance of 2 hops, versus adaptive bidding of sending bids 0, 1, or 2 hops. As described as part of our algorithm, bids received by a host are adjusted to reflect the cost of transmitting a process. We refer to this as a movement bias. In the current simulations this cost is modeled simply as a function of the number of hops that a process is to move. Future tests will model the cost of transmission as a function of process size and distance.

Table 1 - Arrival Rates (Processes/sec)

HOST NUMBER	LIGHT	MODERATE	HEAVY
1	.1176	.1492	.1667
2	.0952	.1176	.1333
3	.1111	.1429	.1538
4	.0625	.1000	.1538
5	.1250	.1250	.1333

For the simulation results presented here, certain parameters are held fixed. These include the topology (Figure 2), the service times of each host (5, 7, 6, 5 and 7 secs for hosts 1-5, respectively), the process size distribution (Table 2), estimated scheduling algorithm overhead (50 ms per invocation), and the amount of memory at each site (2.5 Mbytes).

Figure 2 - Topology For Simulations

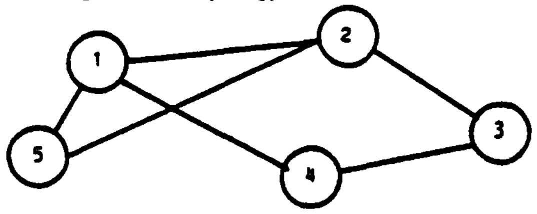


Table 2 - Process Size Distribution (Bytes)

.10	10,000	.85	22,000
.20	12,000	.90	30,000
.40	14,000	.95	34,000
.60	16,000	.98	38,000
.70	18,000	.99	44,000
.8	20,000	.995	50,000

6.0 SIMULATION RESULTS

A significant number of simulation runs were performed that are not described here due to space limitations. These runs were used to determine a reasonable length for the simulations (where they reach steady state) and to choose values for various parameters on the simulation model (see [17]). These values were then used in subsequent runs. A brief description of some of the simulation results for algorithms 1-5 is now presented. Results for algorithm 6 are not yet available. Further, a comparison of the results to simple analytical models and baselines shows the effective operation of the algorithms.

Algorithms 1 and 2

Tables 3 and 4 illustrate the performance of algorithms 1 and 2, respectively under various arrival rates. As expected, percentage of process movement increases and response time decreases as the load gets heavier. In comparison, the percentage of process movement is higher for algorithm 2 because algorithm 2 uses the heuristic of balancing the queue length (which is often not balanced), thereby stimulating more movement in the simulations. Algorithm 1 tries to assign processes to hosts with more free memory. It turned out that memory size was not too limited in these simulations so that this criterion did not induce much movement. This is further illustrated by the unbalanced queue lengths given in Tables 3 and 4. Comparing the response times of these two algorithms indicates that using the amount of free memory by itself (algorithm 2) is not a good variable on which to base scheduling decisions. Under the conditions tested using queue lengths as an indicator of host busyness performs better than using free memory as an indicator.

7 of 11

	LIGHT	MODERATE	HEAVY
System-wide Response Time	20.47	23.50	26.19
Percentage of process movement	19.8	29.6	31.1
Load Balancing			
Host 1	2.419	5.083	5.221
Host 2	1.934	4.734	4.793
Host 3	0.643	0.902	2.129
Host 4	0.089	0.322	1.131
Host 5	1.289	0.919	1.992

Movement Bias	= 4000/hop
Scheduling Interval	= 2000 msec
Subnet Delay	= 200 msec/packet/hop

Table 3. Variation of Arrival Rates (Algorithm 1)

	LIGHT	MODERATE	HEAVY
System-wide Response Time	15.28	18.89	37.02
Percentage of process movement	37.7	43.5	51.4
Load Balancing			
Host 1	0.819	1.940	4.963
Host 2	0.850	1.612	4.921
Host 3	0.641	1.440	4.700
Host 4	0.345	1.288	4.537
Host 5	0.711	1.226	4.049

Movement Bias	= 4000/hop
Scheduling Interval	= 2000 msec
Subnet Delay	= 200 msec/packet/hop

Table 4. Variation of Arrival Rates (Algorithm 2)

Algorithm 3

Algorithm 3 combines algorithms 1 and 2 in an appropriate proportion, determined by simulation (Table 5). The response time for the algorithm 3 was lowest when the criterion was (A1+A2)/2. In all subsequent tests of algorithm 3, (A1+A2)/2 was used as the criterion. These tests indicate the normalization process required by our approach. Of course, this approach is much more complicated when many factors are involved.

Algorithm 3 was then run for various arrival rates and it performed better than algorithm 1 or 2 alone (Table 6), across all arrival rates. This result shows that normalizing these two simple factors can be accomplished to the overall benefit of the network.

Additional simulations were then run to determine the effect of various scheduling intervals (Table 7) and subnet delays (Table 8). Scheduling intervals of between 2 and 4 seconds had the same affect on response time implying that there is no need to run the scheduler at 2 second intervals. However, running the scheduler every second still produced an improvement in response time, and since the cost of running the scheduler is incorporated into the simulation, it does make sense to run the scheduler that frequently. Presumably, at faster and faster periods for the scheduler there will be some point at which the overhead is costlier than the achieved benefit. This was indeed observed for other simulations [21] but not tested for the particular set of parameters presented here.

When testing various sunbet delays, response time deteriorated and percentage of movement increased as the delay in the subnet increased. The degradation in response time is due to longer transit times and more outdated state information as delays increase. The increase in percentage of movement is due to the fact that processes in transit do not alter the level of busyness at the receiving host for longer and longer times as the delays increase, causing even more processes to be transmitted. A possible solution to this problem can be found in [21].

	A1	$\frac{3*A1+A2}{4}$	$\frac{A1+A2}{2}$	$\frac{A1+3*A2}{4}$	A2
System-wide Resp.Time	23.50	17.24	15.14	16.70	18.89
Percent. of movement	29.6	28.8	31.8	39.0	43.5
Load Bal.					
Host 1	5.083	1.740	1.245	1.308	1.940
Host 2	4.734	1.701	1.222	1.313	1.612
Host 3	0.902	1.257	0.898	1.267	1.440
Host 4	0.322	0.810	0.637	1.074	1.288
Host 5	0.919	1.327	0.890	1.099	1.226

Algorithm	= 1 and 2
Load	= Moderate
Movement Bias	= 4000/hop
Scheduling Interval	= 2000 msec
Subnet Delay	= 200 msec/packet/hop

Table 5. Tuning of Algorithm 3.

8 of 11

	LIGHT	MODERATE	HEAVY
System-wide Response Time	12.80	15.14	24.23
Percentage of process movement	24.2	31.8	25.3
Load Balancing			
Host 1	0.606	1.245	3.500
Host 2	0.607	1.222	3.315
Host 3	0.572	0.898	2.857
Host 4	0.230	0.637	2.403
Host 5	0.635	0.890	2.618

Movement Bias	= 4000/hop
Scheduling Interval	= 2000 msec
Subnet Delay	= 200 msec/packet/hop

Table 6. Variation of Arrival Rates (Algorithm 3)

	Scheduling Interval (msec)			
	1000	2000	4000	8000
System-wide Response Time	13.81	15.14	15.27	18.33
Percentage of process movement	31.3	31.8	23.1	21.6
Load Balancing				
Host 1	0.702	1.245	0.979	1.819
Host 2	0.680	1.222	1.092	1.714
Host 3	0.574	0.898	0.774	1.635
Host 4	0.461	0.637	0.403	1.304
Host 5	0.464	0.890	0.896	1.829

Load	= Moderate
Movement Bias	= 4000/hop
Subnet Delay	= 200 msec/packet/hop

Table 7. Variation of Scheduling Interval (Algorithm 3)

	Subnet delay (msec/packet/hop)			
	50	200	400	800
System-wide Response Time	12.78	15.14	18.92	23.54
Percentage of process movement	27.6	31.8	29.5	38.3
Load Balancing				
Host 1	0.979	1.245	1.565	1.756
Host 2	1.086	1.222	1.928	1.671
Host 3	0.739	0.898	2.070	1.301
Host 4	0.609	0.637	1.039	0.728
Host 5	0.781	0.890	1.150	1.002

Load	= Moderate
Movement Bias	= 400/hop
Scheduling Interval	= 2000 msec

Table 8. Variation of Subnet Delay (Algorithm 3)

Algorithm 3 was also tested under moderate load for four different variations on the distance a bid travels. We fixed the distance at n=0 (no bids), n=1, n=2, and n=adaptive (varies between n=0 and n=2 using the criteria discussed in the description of our algorithm). The results are presented in Table 9 and show that even in a small network, adaptive bidding is effective. For example, there is a 6.7% improvement of the adaptive case over the fixed alternative of n=2 and a 14.6% improvement over the fixed case n=1. Future tests will study larger networks.

Table 9: Adaptive Distance For Bid Requests

	n=0	n=1	n=2	n=adaptive
Response Time (sec)	26.31	17.73	16.24	15.14

A simple analytical upper bound can be calculated for each set of arrival rates by treating each host as a M/M/1 queue and averaging the results. An analytical lower bound can be calculated by assuming that the service rate, μ_i is ordered, so that $\mu_i \geq \mu_j$ iff $i<j$ and processes always use the fastest host available. Further, processes will instantaneously shift to faster hosts upon any process completions at a faster host. The formulas for this lower bound are given in [21]. All analytical results are expressed in the Table below with a summary of the response times for algorithms 1, 2 and 3.

9 of 11

SUMMARY OF RESPONSE TIMES (SEC)

Arrival Rate	Analytical Upper Bnd	A1	A2	A3	Analyt. Lower Bnd
LIGHT	28.76	20.47	15.28	12.80	11.91
MODERATE	38.83	23.50	18.89	15.14	12.79
HEAVY	72.72	26.19	37.02	24.43	15.36

These statistics show the effective operation of even these simple bidding algorithms, especially A3.

Algorithm 4

Simulations to test algorithm 4 and 5 used arrival rates 0.1143, 0.0941, 0.1143, 0.08 and 0.1 processes per second for hosts 1-5 respectively. Additional processes which had distributed group characteristics were also inserted into the simulation. The distributed groups were of size 3, 4 or 5 with equal probability and were generated on the average every 20 secs. In changing the arrival rates like this we obtain similar (although not exact) overall arrival rates to the moderate loads used in the simulations of Algorithms 1-3. However, the required run time of a distributed group process is much greater than a normal process due to the penalty we impose (see the description of algorithm 4 in section 4). This degrades overall response time in comparison to Algorithms 1-3. Out of the total number of processes in the system, 28.5% were part of a distributed group. A baseline where the scheduler did not use distributed group information (that is, the McCulloch Pitts neuron did not alter its bid based on this information - in effect, algorithm 3) and a full simulation of algorithm 4 were run.

The results in Table 10 show that only a 5% improvement in overall response time was accomplished by using distributed group information. Notice that the new distributed group processes increased the response time from 15.14 sec (from algorithm 3 results, Table 8) to 26.93 sec (Table 10). When algorithm 4 was run the response time improved to 25.45 sec. Thus the added response time (26.93 - 15.14) was cut by 12.6%. In any case the improvement may be significant because only 28.5% of the total processes were members of distributed groups. Of course, the results are also a function of movement costs, host loads, and how costly it is to <u>not</u> separate distributed group members. More tests are needed to establish the conditions under which using distributed group information causes significant improvement in response time.

	BASELINE (Without MP Neuron)	With MP Neuron
System-wide Response Time	26.93	25.45
Percentage of process movement	32.4	30.7

Algorithm	= 4
Load	= see text of paper
Movement Bias	= 4000/hop
Scheduling Interval	= 2000 msec
Subnet Delay	= 200 msec/packet/hop

% of processes belonging to distributed groups = 28.5%

Table 10. Performance with Distributed Groups

Algorithm 5

The simulations for algorithm 5 were run in a similar manner to algorithm 4 except the additional processes (again groups of 3, 4 or 5 processes) were members of clusters and not distributed groups (Table 11). In this test the use of cluster information improves overall response time by about 5% or if we compare the improvement with respect to added response time over algorithm 3 we get a 10.2% improvement. However, clusters cause more of a degradation in overall response time than do distributed groups (e.g., 30.16 sec for clusters versus 26.93 sec for distributed groups in the two baselines). This is because formation of clusters at a site (the goal) still increases the load at those sites resulting in less balanced queues and slower response times. Again, many more combinations of parameters must be tried before we can generalize these results.

	BASELINE (Without MP Neuron)	With MP Neuron
System-wide Response Time	30.16	28.64
Percentage of process movement	28.4	31.3

Algorithm	= 5
Load	= see text of paper
Movement Bias	= 4000/hop
Scheduling Interval	= 2000 msec
Subnet Delay	= 200 msec/packet/hop

% of processes belonging to cluster groups = 29.0%

Table 11. Performance with Cluster Groups

7.0 CONCLUSIONS

Decentralized process scheduling using many factors is a complicated issue. Which factors have significant impact on performance (response time), which do not, how do various factors interrelate, and how are the three questions just posed are answered under various system conditions are important but difficult questions. We have developed a decentralized scheduling algorithm based on bidding in which it is easy to vary the factors and their interrelationships. A simulation model has been implemented to test the variations of the algorithm under any number of system conditions. Simulation results can be compared to simple analytical models and baseline simulations, as we did, or where, for example, 100% accurate data is used instead of delayed information. Our current simulation results indicate that (under the conditions tested), (1) bidding is quite effective even with simple parameters, (2) the use of cluster and distributed group information in bidding is only marginally worthwhile (but its importance is expected to increase as the percentage of processes exhibiting cluster and group characteristics increase), and (3) transmitting bids a distance which is a function of the system load is effective. Of course, with such a complicated simulation model the simulation results have to be considered preliminary until a much wider set of conditions and combination of parameters are tested. Such tests are currently being performed.

8.0 ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under grant MCS-8104203 and by the US Army CECOM, CENCOMS under grant DAAB07-82-K-J015.

9.0 REFERENCES

1. Agrawala, A. K., S. K. Tripathi, and G. Ricart, "Adaptive Routing Using Virtual Waiting Time Technique," IEEE Transactions on Software Engineering, Vol. SE-8, No. 1, January 1982.

2. Bokhari, S. H., "Dual Processor Scheduling with Dynamic Reassignment", IEEE Transactions on Software Engineering, Vol. SE-5, No. 4, July 1979.

3. Casey, L. and N. Shelness, "A Domain Structure for Distributed Computer Systems", Proceedings of Sixth ACM Symposium on Operating System Principles, November 1977.

4. Chou, T. C. K. and J. A. Abraham, "Distributed Control of Computer Systems", Technical Report, Coordinated Science Laboratory, University of Illinois, 1981.

5. Chou, T. C. K. and J. A. Abraham, "Load Balancing in Distributed Systems", IEEE Transactions on Software Engineering, Vol. SE-8, No. 4, July 1982.

6. Chow, Y. C. and W. H. Kohler, "Models for Dynamic Load Balancing in Heterogeneous Multiple Processor Systems, IEEE Transactions on Computers, Vol. C-28, No. 5, May 1979.

7. Chu, W. W., L. H. Holloway, M. Lan and K. Efe, "Task Allocation in Distributed Data Processing", IEEE Computer, Vol. C-18, November 1980.

8. Efe K., "Heuristic Models of Task Assignment Scheduling in Distributed Systems", IEEE Computer, Vol. 15, No. 6, June 1982.

9. Enslow, P., "What is a Distributed Data Processing System?", IEEE Computer, Vol. 11, No. 1, January 1978.

10. Ezzat, A. K., "Decentralized Control of Distributed Processing Systems", PhD Dissertation, Department of Electrical and Computer Engineering, University of New Hampshire, December 1982.

11. Farber, D. J. and K. C. Larson, "The Structure of a Distributed Processing System - Software", Proceedings of the Symposium on Computer Communication Networks and Teletraffic, Polytechnic Institute of Brooklyn, April 1972.

12. Farber, D. J., et al, "The Distributed Computer System", Proceedings of the Seventh Annual IEEE Computer Society International Conference, February 1973.

13. Gylys, V. B. and J. A. Edwards, "Optimal Partitioning of Workloads for Distributed Systems", Proceedings COMPCON, Fall 1976.

14. Jones, A. K., et al. "StarOS, A Multiprocessor Operating System for the Support of Task Forces," Proc. 7th Symposium on Operating System Principles, 1979.

15. Ma, P. Y. R., E. Y. S. Lee and M. Tsuchiya, "A Task Allocation Model for Distributed Computer Systems", IEEE Transactions on Computers, Vol. C-31, No. 1, January 1982.

16. Ousterhout, J. K., D. A. Scelza and P. S. Sindhu, "Medusa: An Experiment in Distributed Operating System Structure", Communications of the ACM, Vol. 23, No. 2, February 1980.

17. Sidhu, I. S., "Decentralized Process Scheduling in a Distributed Computing Environment", Masters Thesis, University of Massachusetts, Amherst, July 1983.

18. Smith, R. G., "The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver", IEEE Transactions

11 of 11

on Computers, Vol. C-29, No. 12, December 1980.

19. Soloman, M. H., and R. A. Finkel, "Roscoe: A Multi-microcomputer Operating System," Proc. 2nd Rocky Mountain Symposium on Microcomputers, August 1978.

20. Stankovic, J. A., "The Analysis of Decentralized Control Algorithms Using Bayesian Decision Theory", Proceedings 1981 International Conference on Parallel Processing, August 1981.

21. Stankovic, J. A., "Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms", Computer Networks, to appear.

22. Stankovic, J. A., "Bayesian Decision Theory and Its Application to Decentralized Control of Job Scheduling," submitted to IEEE Transactions on Computers, May 1983.

23. Stone, H. S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms", IEEE Transactions on Software Engineering, Vol. SE-3, January 1977.

24. Stone, H. S. and S. H. Bokhari, "Control of Distributed Processes", IEEE Computer, Vol. 11, No. 7, July 1978.

25. Wittie, L. and André van Tilborg, "MICROS, A Distributed Operating System for Micronet, A Reconfigurable Network Computer," IEEE Transactions on Computers, Vol. C-29, No. 12, December 1980.

DYNAMIC TASK SCHEDULING IN DISTRIBUTED HARD REAL-TIME SYSTEMS

Krithivasan Ramamritham
Computer and Information Science

John A. Stankovic
Electrical and Computer Engineering

University of Massachusetts
Amherst MA 01003

ABSTRACT

Most systems that are required to operate under severe real-time constraints assume all tasks and their characteristics are known a priori. Scheduling of such tasks can then be done statically. Further, scheduling algorithms under such conditions are usually limited to multiprocessor configurations. This paper presents a scheduling algorithm, that works dynamically and on loosely coupled distributed systems, for tasks with hard real-time constraints, i.e., the tasks must meet their deadlines. It contains a local guarantee routine and a network wide bidding routine that is specifically suited to hard real-time constraints and other timing considerations. Periodic tasks, non-periodic tasks, scheduling and communication overheads, and preemption are all accounted for in the algorithm. Simulation studies are used to evaluate the performance of the algorithm.

This work was supported in part by the US Army CECOM, CENCOMS under grant DAAB07-82-K-J015 and by the NSF under grant MCS 82-02586.

1.0 INTRODUCTION

Many tasks performed in systems such as those found in nuclear power plants and automated flight control are inherently distributed and have severe real-time constraints. These tasks have execution deadlines that must be met and are thus said to have hard real-time constraints. With today's advances in software, hardware and communication technology for distributed systems, it may be possible to deal with distributed real-time systems in a more flexible manner than in the past. One of the challenges in designing such systems lies in the scheduling of tasks such that the tasks meet their deadlines. Most current research on scheduling tasks with hard real-time constraints is restricted to multiprocessing systems and hence are inappropriate for distributed systems. In addition, many of the proposed algorithms assume that all tasks and their characteristics are known in advance and hence are designed for static scheduling. Our research is directed at developing task scheduling software for loosely-coupled systems with the goal of achieving flexibility through the dynamic scheduling of tasks in a distributed and adaptive manner. Our results include the development of a locally executed guarantee algorithm for periodic and nonperiodic tasks, the creation of a network-wide bidding algorithm suited to real-time constraints, the possibility of preemption when it is determined that it will not cause missed dealines, and inclusion of different types of overheads, including running some of the scheduling tasks as periodic tasks.

Our scheme for distributed dynamic scheduling requires the presence of one scheduler per node. These schedulers interact in order to determine where a newly arriving task could be scheduled. Associated with a node in a distributed system is a set, possibly null, of periodic tasks which are guaranteed to execute on that node. We assume that the characteristics of periodic tasks are known in advance and that such tasks must meet their deadlines. At system initialization time it is verified that there is enough processing power at a node for all periodic tasks to meet their deadlines. In addition to the periodic tasks, we allow for the arrival of nonperiodic tasks at any time and attempt to guarantee these tasks dynamically, in the presence of periodic tasks and on a network wide basis.

This paper is organized as follows. Section 2 outlines some of the current work in guaranteeing tasks in hard real-time systems. Section 3 discusses the nature of real-time tasks as well as the constraints attached to the execution of these tasks. The structure of the scheduler on a node is presented in section 4 while the interactions among the schedulers are the subject of section 5. Since our scheduling algorithm is known to be non-optimal, it is necessary to evaluate its performance. An evaluation plan and preliminary results of simulation studies performed on our scheduling algorithm are presented in section 6. Section 7 summarizes the significant features of our approach to distributed task scheduling.

2.0 BACKGROUND

Most research on scheduling tasks with hard real-time constraints is restricted to uniprocessor and multiprocessor systems. For example, Garey and Johnson [GARE75] describe an algorithm to determine if a two processor schedule exists so that all tasks are completed in time, given a set of tasks, their deadlines, and the precedence constraints of all tasks; Liu and Layland [LIU73] derive necessary and sufficient conditions for scheduling periodic tasks, with preemption permitted. Their results, which hold for uniprocessor systems, were subsequently extended to include arbitrary task sets [DERT74] and precedence constraints [HENN78]. Teixeira [TEIX78] develops a model that considers priority scheduling. It addresses external interrupts, scheduling overhead and preemption. Johnson and Madison [JOHN74] develop a measure of free time, similar to our notion of surplus processing power, to determine whether new jobs can be permitted to execute on multiprocessor systems. These schemes are all quite inflexible and are restricted to uniprocessor and multiprocessor systems. We are attempting to develop more flexible dynamic scheduling techniques for loosely-coupled networks.

Muntz and Coffman [MUNT70] have developed an efficient algorithm to determine minimal length preemptive schedules for tree-structured computations in multiprocesor systems. Similarly, Leinbaugh has developed analysis algorithms which when given the device and resource requirements of each task and the cost of performing system functions determines an upper bound on the response time of each task. His initial results [LEIN80] for multiprocessor systems have been extended to distributed systems in [LEIN82]. Resource requirements and operating system overheads are accounted for in these papers but periodic tasks are not. While these approaches are useful at system design time to statically determine the upper bounds on response times, there is no attempt at guaranteeing that a new task will meet its deadline, although it seems possible to extend them in this manner.

Multiprocessor scheduling in a hard real-time environment has been described by Mok and Dertouzos [MOK78] by a scheduling game in which tokens are moved on a coordinate system defined by laxity and computation time. They have obtained the following results, which we use in our approach:

1. In the case of a single processor, both earliest deadline scheduling (scheduling the task with the earliest deadline) and least laxity scheduling (scheduling the task with the least difference between its deadline and computation time) are optimal.
2. In the multiprocessor case, neither is optimal.
3. For two or more processors, no scheduling algorithm can be optimal without a priori knowledge of (i) deadlines (ii) computation times, and (iii) start-times of the tasks.

Finally, it is known that optimal scheduling in a multiprocessing environment is an NP-hard problem and is hence computationally intractable [GRAH79]. The loosely-coupled nature of distributed systems makes the problem even harder. Clearly then, a practical scheduling algorithm has to be based on heuristics in order to reduce scheduling costs and has to be adaptive. This is the context in which we have been studying the problem of scheduling in distributed systems.

3.0 NATURE OF TASKS

Our research involves scheduling tasks, with real-time constraints, on processors in a network. A task is characterized by its start time, computation time, deadline and possibly its period. In addition, a task may have resource requirements and precedence constraints. In this paper we confine our attention to the scheduling of tasks taking only deadlines into consideration. The extension of our algorithm to handle resource and precedence requirements is currently being studied.

Tasks may be periodic or nonperiodic. A nonperiodic task is one that occurs in the system just once and at unpredictable times. Upon arrival it is characterized by its deadline and its computation time. Such a task can be scheduled any time after its arrival. A periodic task, say with period P, is one that has to be executed once every P time units, i.e., there should be one execution of the task every P units. We do not associate any other semantics with periodic tasks (such as, P time units should elapse between two consecutive executions of a periodic task). In a real-time system there could be a number of such periodic tasks, with different periods. The initial distribution of periodic tasks is assumed known, although our simulation model can also be used as a tool in choosing a good assignment of periodic tasks to hosts in a network.

From a scheduler's point of view, a periodic task represents tasks with known (future) start times and deadlines, whereas, a nonperiodic task may arrive at any time, can be started anytime after its arrival, and may have arbitrary deadlines. In both cases, we assume that a task's computation time is known a priori. While we do not do so, our algorithm can also easily handle non-periodic tasks with future start times.

4.0 THE STRUCTURE AND FUNCTIONING OF THE SCHEDULER ON A NODE

Each node in the distributed system has a scheduler local to that node. A set (possibly null) of guaranteed periodic tasks exist at each node. Nonperiodic tasks may arrive at any node in the network. When a new task arrives, an attempt will be made to schedule the task at that node. If this is not possible, then the scheduler on the node interacts with the schedulers on other nodes, using a bidding scheme, in order to determine the node on which the task can be sent to be scheduled. Upon arrival at the destination node another attempt is made to schedule the task. Eventually, the task either gets guaranteed and executed, or is not guaranteed.

Underlying our scheduling algorithm is the notion of <u>guarantee</u>ing a task. A task is said to be guaranteed if under all circumstances it will be scheduled to meet its real-time requirements. Thus, once a task has been guaranteed, all that is known is that the execution of the task will be completed before its deadline; exactly when it will be scheduled is dependent on the scheduling policy, the tasks that have been guaranteed but are waiting to be executed, the nature of periodic tasks, new arrivals, and the amount of system overhead.

While periodic tasks are guaranteed tasks, nonperiodic tasks, after they arrive, may or may not be guaranteed. However, once guaranteed, they will definitely meet their deadlines. The intent then is to guarantee all periodic tasks and as many of the nonperiodic tasks as possible, utilizing the resources of the entire network.

The following figure shows how the various modules, namely the local scheduler, the dispatcher, and bidder as well as the guarantee subroutine, that make up the scheduler on a node interact with each other. Their functions are discussed next.

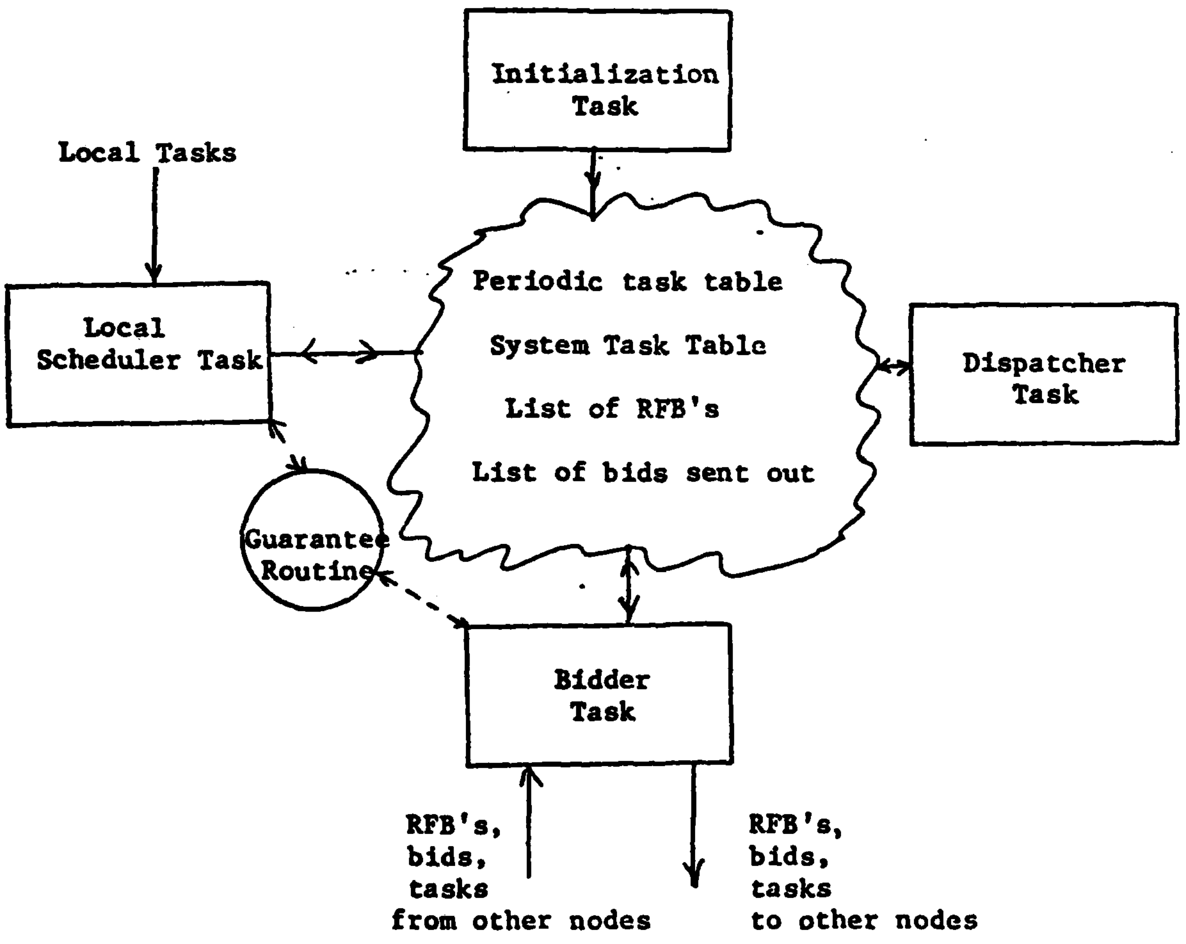


Figure 1: Structure of a scheduler

Bidder and Local Scheduler Tasks

Tasks may arrive directly at a node or as a result of a bidding process. Tasks in the former category are handled by the Local Scheduler task while tasks in the latter are handled by the Bidder task. Arriving tasks may or may not cause preemption. The conditions for both are derived below. Since the bidder is involved in the distributed aspect of task scheduling, it is discussed in the next section.

The local scheduler first calls the guarantee routine in order to guarantee the new task. If it is guaranteed, then the dispatcher is invoked. Otherwise, the task is placed in the bidder's queue of tasks for which bids have to be requested from other nodes and then the dispatcher is invoked.

The Dispatcher Task:

It is the dispatcher task that determines which of the guaranteed periodic and nonperiodic tasks is to be executed next. As mentioned earlier, for a uniprocessor, both the earliest deadline algorithm and the least laxity algorithms are optimal. In our scheme, guaranteed tasks are executed according to the earliest-deadline-first scheme. It should be mentioned that our use of the earliest deadline algorithm for scheduling tasks on a single node does not guarantee an optimal schedule on the network as a whole, which is a computationally infeasible problem. Our aim here is to guarantee tasks quickly and to reduce overheads. The simulation studies are an attempt to analyze the algorithm's behavior.

The dispatcher's actions are simple: whenever, a task completes, the dispatcher is invoked and it selects the task with the earliest deadline for execution. To expedite this selection, the list of guaranteed tasks is ordered according to the deadlines of the tasks. The runtime cost of the dispatcher is part of the computation time of every task.

For the purpose of scheduling, a "template" for periodic tasks is maintained in a data structure called the periodic task table (PTT). During the operation of the system the guarantee algorithm uses the template in conjunction with the concept of "surplus" to ascertain whether non-periodic tasks can be guaranteed. Surplus is derived from information in the system task table (STT). Both STT and the surplus are described below.

<u>The System Task Table</u>:

Each node maintains a STT for all local periodic and non-periodic tasks guaranteed at any point in time. Each entry in the STT contains an arrival time, a latest start time, a deadline and computation time. All but the latest start time are inputs. Tasks in the STT are ordered by their arrival times and within each arrival time by deadlines.

To compute the latest start time of a task with deadline D, all guaranted tasks with deadlines greater than or equal to D are ordered according to decreasing deadlines. The latest start time is determined by assuming that tasks are scheduled to execute just in time to meet their deadlines. For example, if the first task on the list has deadline D1 and computation time C1, it has a latest start time of D1-C1. Suppose the second task on the list has a computation time C2 and deadline D2. If D2 is greater than D1-C1 then the task has a latest start time of D1-C1-C2, otherwise D2-C2. In this manner, latest start times are calculated for every guaranteed task. This is, in effect, the reverse of the Mok and Dertouzos' token scheduling game [MOK78].

Surplus

It should be clear that only if the surplus processing power at a node is greater than the computation time requirement for a newly arriving task will it be possible for the task to be guaranteed to execute at that node. Thus, we are interested in surplus with respect to the task about to be guaranteed or rejected. Surplus, then, is defined as the amount of computation time available on a node between the time of arrival of the new unguaranteed task and its deadline. Tasks that arrive locally have arrival time equal to zero; Tasks under consideration in the bidding process have an estimated non-zero arrival time.

While surplus is not explicitly calculated for guaranteeing local tasks, surplus information is taken into account during the computation of the latest start time for such tasks: only if the latest start time is greater than the arrival time of the task is the task guaranteed. Surplus is computed explicitly while responding to a request for bid.

The Initialization Task:

Before the guarantee routine examines newly arriving tasks, its data structures have to be initialized. This is the function of the Initialization Task which is executed just once, at system initialization time. Since a periodic task is known to represent regularly occurring tasks with future start times, at system initialization time it has to be determined if there is enough processing power to execute a given set of periodic tasks on a node. We define a <u>window</u> to be the least common multiple, L, of the periods P1,...,PN of all periodic tasks 1,..,N. The tasks have computation time C1,...CN. A necessary condition for periodic tasks in a window to be guaranteed is

$$\text{If} \quad \sum_{i=1}^{N} \{Ci * (L/Pi)\} \quad <= L.$$

that is, the sum of the computation times of all periodic task instances that have to be executed within the window is less than or equal to the window itself. After this necessary condition is verified, the latest start time for every task instance in the window is calculated with an algorithm described previously. If all the tasks have latest start times no earlier than their arrival times, they are guaranteed. The calculated start times, computation times, and deadlines are then stored in the PTT for each task instance in the window.

<u>The guarantee Routine</u>:

The Guarantee Routine local to a node is invoked to determine if there is enough surplus processing power to execute a newly arriving task before its deadline. A task can be guaranteed only after ascertaining that guaranteeing the task does not jeopardize not only previously guaranteed nonperiodic tasks but also periodic tasks with future start times. If a newly arriving task cannot be guaranteed locally, the task becomes a candidate for bidding (see section 5). As described by the following algorithm, the guarantee routine uses information in the PTT and the STT to guarantee a newly arriving task.

Recall that each entry in the STT contains an arrival time, a latest start time, a deadline and computation time.

<u>Outline of the Guarantee routine</u>:

```
Begin
        Let 'tick' be the current clock time;

        Let the new task be a 4-tuple, (A,S,D,C), where
                A = arrival time, S = latest start time,
                D = deadline,     C = computation time;

        Let lastD be the latest deadline of the last window
        in the current STT;

        Do while (lastD < D)
                Append a copy of PTT to the end of STT; add lastD to
                all the clock times of tasks in the appended copy of
                PTT; Update lastD;
        end do

        Find the current entry, i, in STT, such that (A < Ai) or
        (A = Ai and D < Di);

        Find the latest start time, S, for the new task taking into
        account tasks from the current entry i to the end of the STT;

        If (S < tick ) or (S < A)
        then    return ('not guaranteed');
        else
                Temporaily consider (A,S,D,C) as inserted prior to
                the entry i and be the new current entry;

                For each entry j from the current entry backwards
                    to the first entry,
                Do
                    Calculate new latest start time, Sj;
                    If (Sj < tick) or (Sj < Aj) then
                        new task cannot be guaranteed;
                        return ('not guaranteed');
                    else
                        temporarily save Sj;
                    endif
                end do

                /*New task is guaranteed */
                Commit the insertion of (A,S,D,C) and all the new
                start times Sj calculated in the above loop;

                If this is a task request as part of a request for bid
                then
                        Add up the CPU idle time between A and D,
                        and return this value (surplus) to the
                    invoking process;
                else
                    return ('guaranteed');
                endif
        endif
end;
```

Consideration of time overheads in scheduling:

One of the prime motivations for the above separation of scheduling activity among various scheduling tasks is to take into account time spent on scheduling. This is important in hard real-time systems.

Since the dispatcher has to be invoked each time any task, including the local scheduler task and the bidder task, completes execution and relinquishes the CPU, we require that the time taken for the dispatcher to execute be included in the computation time of every task.

It is essential that newly arriving nonperiodic tasks be examined soon after they arrive. But, interrupting a running task to guarantee a newly arriving task might result in the running task missing its deadline. The following scheme is utilized in order to solve this problem: the worse-case computation times for the bidder task and for the local scheduler task are determined. After the dispatcher chooses the next task to run, using the SST, it checks if there is sufficient surplus such that running the bidder task or the local scheduler task, after preempting the newly dispatched task, does not result in guaranteed tasks missing their deadlines. If the above is true for the bidder (local scheduler) task, then the dispatcher sets the Invoke bidder bit (Invoke local scheduler bit).

Even if a running task is not preemptable, thereby preventing a new task from being guaranteed soon after its arrival, the newly arriving task should be examined without undue delay. To facilitate this, both the bidder and the local scheduler tasks are executed as periodic tasks. The period and computation time of these tasks is a priori determined by the nature of tasks, for instance, the estimates of their laxity and frequency of arrival, as well as on the nature of communication from other nodes in the system, for instance, the frequency of request for bids.

The above scheme is based on the assumption that there is a communication module, executing on a processor which is separate from the CPU on which tasks are scheduled, that is responsible for receiving communication from local sources as well as from other nodes. Based on the type of communication, this module stores received information in the appropriate data structures so that they will be looked at when the different tasks execute. In addition, if a task arrives from another node and the invoke bidder bit has been enabled, then the currently running task is preempted and the bidder task is executed. If instead, a task arrives locally and the invoke local scheduler bit is set, then the currently running task is preempted and the local scheduler task is executed.

In a logical extension to the above scheme, wherein a separate processor is allocated for the communication module, we have been studying an architecture for a real-time distributed system wherein a separate processor, with limited processing capabilities, is allocated for the purposes of scheduling. In this scheme, scheduling overheads are incurred only by the additional processor since the main processing

It is necessary to take communication delays into account during the bidding process. It should also be recognized that communication delays depend on the pairs of processes involved, for instance, on the distance separating them and on the communication from other nodes in the system to the two nodes, etc. We plan to estimate the time in the following way: every communication will be timestamped by the sending node. The receiving node would then be able to compute the delay due to that communication by subtracting the timestamp from the time of receipt. Subsequent communication delays are estimated based on a linear relationship between message length and communication delay. By utilizing the latest known delay, this computation can adapt to changing system loads. Of course, it is necessary for the clocks on different nodes to be approximately synchronized. (See [LAMP78] for schemes to synchronize remote clocks.)

We now describe the different phases of the bidding process in detail. The bidder first checks if it can send the task to another node via focussed addressing. This utilizes surplus information to reduce overheads in the bidding process and works as follows: Using the surplus information about nodes, if it can be determined that a particular node has a surplus which is significantly greater than the computation time of the task and hence has a high probability of guaranteeing the new task, then the new task can be sent directly to that node without bidding. We propose to do this in the following manner: Estimate the transit time T for the task to reach the selected node. If the estimated surplus of that node, between T and the deadline D of the task, is greater than the computation time C of the task by FP percent, then the task is sent to the node. The receiving node, as stated earlier, uses the guarantee routine to check if the arriving task can be guaranteed there. FP is an adaptive parameter used in focussed addressing.

If there is no node with a significant amount of surplus, then bidding is resorted to. In this case, the main functions of the bidder are: sending out request for bids for a task that cannot be guaranteed locally, responding to bids, and responding to the request for bids from other nodes. These functions are detailed now.

Request for Bids: For a task that cannot be locally guaranteed, the bidder broadcasts to all nodes a Request For Bid (RFB) message containing the following: the computation time C of the task, the deadline D of the task, the size S of the task, the time T at which the message is being sent, and a deadline for responses R. R is the time after which the requesting process will examine the bids to choose the best bidder and should be such that after R there is sufficient time

1. for the requesting process to evaluate the bids,
2. for the task to reach the best bidder node,
3. for the best bidder to guarantee and schedule the task, and
4. once scheduled, for the task to complete computations and meet its deadline.

Thus R is equal to

$$D - (P + E + W + C)$$

where P is the period of the task that evaluates bids and hence the maximum waiting time before bids are recognized; E is the estimate of the average time taken for the task to reach the best bidder; W is a window, being the estimate for how long, after arrival, the task might begin computation. Both E and W are tuned to adapt to the load on the communication network as well as to the surplus of the receiving node.

If after R is computed it is estimated that there isn't sufficient time for the requests to reach other nodes and for them to send their bids, then the bidder resorts to focussed addressing instead of bidding. In this case, FP, the adaptive parameter used in focussed addressing is adjusted so that the chances of finding a node with surplus are greater. If still a node with surplus cannot be found, the task cannot be guaranteed.

A possible improvement over the above scheme for requesting bids, one in which RFB's are broadcast to all nodes, is to send the RFB's only to nodes whose estimated surplus matches the requirements of the task that needs to be guaranteed. While this has the advantage of avoiding potentially unnecessary communication, time is taken to determine potential bidders from the knowledge of the surplus of other nodes. Also, if the available surplus information is not accurate, there is a possibility of preventing nodes with surplus from bidding.

<u>Bidding in response to Request for Bids</u>: The bidder first estimates that its response will reach the requestor before the deadline for response. It proceeds with further actions on the request for bid only if the time of response plus the transit time for the response is less than the indicated deadline for response.

Once a node decides to respond, it first computes ART, the estimated arrival time for the task, if indeed it is awarded the task. ART is one of the three components of the bid. Computation of ART takes into account the following: 1) bids at the requesting node are evaluated after the deadline for responses to bids, 2) the average delay in evaluating bids (this is estimated to be half the period of the bidding task), and 3) the estimated time taken for the task to arrive at the bidders's node.

Whether or not the bidder will be in a position to execute the new task is determined by SARTD, the surplus at the bidder's node between ART and D, the deadline of the task. The surplus information takes into account

1. processing time needed for previously guaranteed tasks, including future instances of periodic tasks: this ensures that guaranteed tasks are not jeopardized.
2. processing time needed for tasks that may arrive as a result of previous bids: this ensures that nodes requesting bids are aware of other bids by a node and hence minimizes the probability of a node being awarded tasks with conflicting requirements or being awarded too many tasks creating an unstable situation.
3. processing time needed for nonperiodic tasks that may arrive locally in the future: this minimizes the probability of a task arriving as a result of a bid not being guaranteed due to the arrival of a local task with similar real-time requirements.

While accurate information is available concerning (1), information needed for (2) and (3) is estimated based on the past behavior of the node. In our case, we keep track of PNL, the percentage of CPU time used by nonperiodic tasks arriving locally, and PNB, the precentage of CPU time used by nonperiodic tasks arriving as a result of bidding. These percentages are maintained for the most recent T time units, where T is an adaptive parameter.

Let S be the surplus between ART and D computed taking into account only tasks already guaranteed. Then SARTD, the surplus between ART and D taking into account all three factors mentioned above, is given by

$$S - (PNL + PNB) * (D - ART)$$

Only if SARTD is greater than the computation time, C, of the task, does the bidder respond to the RFB, otherwise not. SARTD is the second component of the bid.

In the next section we will describe how the requestor of bids uses information in the bids to pick the best bidder. But before we describe the bid processing aspect of the scheduler, we present some of the possible improvements upon the above bidding process.

Suppose a node receiving a RFB utilizes its estimates of the surplus of other nodes and determines that another node has a higher probability of being awarded the task. In this case, communication and computation costs incurred in bidding can be reduced if the node decides not to respond to the RFB. Of course, the accuracy of this decision is dependent on the accuracy of the surplus information possessed by the node.

It is quite possible for a node to have sufficient processing requirements locally that it does not want to respond to RFB's. It could intimate all other nodes that it has no surplus, till say some future time; this information could be used, as indicated under requesting for bids, in choosing nodes to which request for bids should be sent. Another possibility is for a node to temporarily disable the mechanism that recognizes RFB's from other nodes.

Bid Processing: This is carried out by the node that originally sent out the request for bids. A bid processor queues all bids sent in response to a RFB until the corresponding deadline for response R. Once the deadline is past, for each bidder it computes ETA, the estimated time of arrival of the task at the bidder's node. For each bidder it estimates SETAD, the surplus between ETA and D assuming the same rate of surplus indicated by the bidder. SETAD is computed using the following formula:

SETAD = SARTD * (D - ETA)/(D - ART)

The bid processor then chooses the one with the greatest SETAD as the best bidder.

In order for bidders to know the response to their bids, so that they can purge unnecessary information, the bid processor intimates all but the best bidder that their bid was not accepted. To the best bidder, the task is sent. A possible alternative, which avoids the traffic resulting from the responses, is for bidders to time-out after a predetermined interval.

One final note about the information sent on bids: A node utilizes this information to keep track of the surplus in other nodes. This is used, as explained earlier, in focussed addressing and for sending RFB's to nodes with high surplus. In response to a RFB for a task, a bidder sends the estimated arrival time and the estimated surplus between the arrival time and the task's deadline. If a node does not respond to a RFB, it is assumed that it does not have sufficient surplus. (In reality, a node may not have sent its bid because it estimated that its bid will not arrive before the deadline for responses.) Information received via bids is bound to be fragmented and hence a node may, if needed, explicitly request information from other nodes regarding their surplus within a specific interval of time. Whichever scheme is followed for obtaining surplus information, due to the nature of activities in the system, very often such information will be outdated by the time they are used. Hence they should be viewed only as estimates which will be updated when nodes bid in the future.

Response to task award: Once a task is awarded to a node, the awardee node treats it as a task that has arrived locally at the node and takes actions to guarantee it or notify the user process that initiated the task if it cannot be guaranteed.

6.0 EVALUATION OF THE SCHEDULING ALGORITHM

The task scheduling algorithm proposed in this paper consists of two main algorithms: the local guarantee algorithm and the bidding algorithm. The evaluation of these algorithms is being done in 2 stages of simulation. The first stage, which is complete, involved the implementation of the local guarantee algorithm in an event driven simulation model. In stage 2, this model is being extended to handle the bidding algorithm.

This section briefly describes the overall simulation model, the planned evaluation process and the stage 1 simulation results. The simulation program is written in GPSS and FORTRAN and it simulates a (variable) number of nodes connected by a local area network (ethernet). All parameters of the model are easily changed to facilitate evaluation. The model is extensible so that precedence constraints and additional resource requirements (other than the CPU) can be added later.

In the simulation model each host is described by its processing speed, it runs all modules of the scheduling algorithm (local scheduler, dispatcher, and bidder), and it executes periodic as well as nonperiodic tasks. As described in our algorithm, tasks can sometimes be preempted. This facility is also included in the simulation model.

Tasks are characterized by their period (if any), their computation time and their deadline. Specific values for these parameters are generated by GPSS function definitions.

The communication subnet (ethernet) is modelled as an average delay per packet per hop. Various types of information move through the subnet, each experiencing different delays as a function of their size. These include requests for bids (1 packet), bids themselves (a variable number of packets), and the tasks (a variable number of packets).

Arrival rates for the nonperiodic tasks are Poisson and occur independently at each node of the network. The set of periodic tasks for each node must be chosen at the start of the test, but can be easily changed from run to run. The initialization task is executed to insure that periodic tasks can all be guaranteed. If not, the task immediately terminates with an error condition.

The statistics accumulated include node utilization, task response time, node queue length, percentage of nonperiodic tasks that meet and do not meet deadline, percentage of nonperiodic tasks that move to another site and then meet or do not meet their deadline, and the number of tasks moved via focussed addressing and the percentage of these that do or do not meet their deadline. Using a log file it is also possible to determine which tasks were and were not guaranteed.

Our testing determines how these statistics change as we vary the characteristics of the periodic and nonperiodic tasks (computation time and deadline), their periods and arrival rates respectively, the delays in the subnet and the size of the network. Since much of the scheduler runs as a periodic task we can easily determine the effect, say, of running the bidder at various periods.

In addition, we would like to determine which parameters of our algorithm have the greatest effect on performance. For example, determining the usefulness of focussed addressing, or determining the difference between an adaptive or non-adaptive policy in setting a wait time before processing arriving bids, or the effectiveness of various types of information in the bids themselves. The simulation will also be used to study the performance of the various possibilities for improvement discussed in section 5.

The various simulation runs will be compared to each other as well as to a (lower bound) baseline model where 100% accurate data about other hosts' schedules are assumed to be known.

The simulation of the guarantee algorithm on a single host provides some insight into the percentage of tasks guaranteed under various conditions (set of base periodic tasks, arrrival rates, task characteristics, etc.). For example, Table 1 shows the simulation results where computation time (cpu time) and deadline characteristics are varied for a set of 2 periodic tasks.

Table 1

Simulation Results - Guarantee Algorithm

	cpu time	^D deadline	tasks guaranteed	tasks not guaranteed	average cpu utilization
(1)	uniform distr (1-10)	exp distr u=4	65.9%	34.1%	70.1%
(2)	normal distr N(4,1)	same as (1)	81.6%	18.4%	60.9%
(3)	normal distr N(6,1)	same as (1)	64.8%	35.2%	70.2%
(4)	normal distr N(8,1)	same as (1)	51.7%	48.3%	73.7%
(5)	normal distr N(9,1)	same as (1)	45.5%	54.5%	73.9%
(6)	normal distr N(10,1)	same as (1)	39.4%	60.6%	72.9%
(7)	normal distr N(12,1)	same as (1)	29.9%	70.1%	69.5%
(8)	normal distr N(14,1)	same as (1)	22.6%	77.4%	65.5%
(9)	same as (3)	uniform distr (1-25)	87.3%	12.7%	85.0%
(10)	same as (3)	normal distr N(15,4)	87.9%	12.1%	85.2%

The two periodic tasks take 24.8% of the total cpu time. The simulation runs for 4,000 units of time. During this period, 783 instances of periodic tasks are executed. The characteristics of the periodic tasks are:

period	cpu
7	1
19	2

window size = 133

A single arrival rate for non-periodic tasks is utilized and assumed to be Poisson distributed. The mean of the interarrival time is 9 units of time. During the simulation time 495 nonperiodic tasks are actually generated. Six different combinations of computation time and deadline distributions are tested. The various combinations are shown in Table 1. Note that the deadline is assigned to tasks according to the formula:

D = current clock + computation time + D.

The distribution of D is evaluated instead of D.

Based on the results shown in the table, from case 1 to case 8, we conclude that, 1) given the same D distribution, if most of the tasks have lower computation time requirements, the percentage of guaranteed tasks is higher, 2) the cpu utilization is higher when most tasks have higher computation time requirements, but reaches a maximum of 73.9% when the computation time distribution is N(9,1), then goes down as the mean of the distribution is even higher. The percentage of guaranteed tasks is higher for lower computation time requirements because they can more easily be guaranteed. Lower cpu utilization for tasks of lower computation time requirements is possibly due to two factors; 1) even though more tasks are guaranteed, total computation time is still less, and 2) cpu time is fragmented by a larger number of small pieces of tasks, which causes more idle cpu time slots. When the computation time distributions go beyond N(9,1), the cpu utilization drops because too few tasks are guaranteed. Note that, in case 8, the cpu utilization is still more than that in case 2, eventhough the percentage of tasks guaranteed is much less.

In cases 3 and 9, we see that, by changing the D distribution, the amount of the guaranteed tasks and the cpu utilization are improved by 35% and 21%, respectively. In this test, we conclude that, with the same computation time distribution, more tasks are guaranteed if most tasks have more laxity. In case 9 versus case 10, improvement in the percentage of guaranteed tasks is negligible. This is because the percentage of tasks of urgent deadlines, which are not easy to guarantee, are about the same in both cases.

Other similar runs with different sets of periodic tasks have been made. As expected, as a higher percentage of time is allocated to periodic tasks, less and less non-periodic tasks are guaranteed and executed. While stage one of the simulations reveal little about the overall approach, they do show that our simulation program can be used to help identify a 'good' set of periodic tasks for a host, given a known non-periodic arrival rate. Stage 2 of the simulation model is being implemented and should produce results within the coming months which have direct relevance to distributed task scheduling.

7.0 CONCLUSION

In this paper we discussed an approach to scheduling tasks with hard real-time constraints, in a loosely-coupled distributed system. The main features of this algorithm are

1. A distributed scheduling technique; no node has greater importance, as far as scheduling is concerned, than any other.

2. The algorithm is designed to schedule both periodic tasks as well as nonperiodic tasks. The latter may arrive at any time.

3. A bidding approach is used for the selection of nodes on which tasks execute. We have worked out the details of the various phases involved in the bidding process.

4. Overheads involved in scheduling are taken into consideration. This includes the time spent on executing scheduling tasks at a node and the communication delays between nodes.

5. Heuristics are built into the algorithm. This is necessary given the computationally hard nature of the scheduling problem.

6. Simulation studies are being performed in order to evaluate many of the alternatives available.

References

[DERT74] Dertouzos, M., "Control Robotics: The Procedural control of physical procsses", Proc. of the IFIP Congress, 1974.

[GARY75] Garey, M.R. amd Johnson, D.S., "Complexity results for multiprocessor scheduling under resource constraints", SIAM Journal of Computing, 4, 1975.

[GRAH79] Graham, R.L. et al, "Optimization and Approximation in Deterministic Sequencing and Scheduling: a survey", Annals of Discrete Mathematics, 5, 1979, 287-326.

[HENN78] Henn, R., "Antwortzeitgesteuerte Prozesourzuteilung unter Strengen Zeitbedingungen", Computing, Vol 19, 1978.

[John74] Johnson, H.H. and Madison, M.S., "Deadline scheduling for a real-time multiprocessor", NTIS (N76-15843), Springfield, VA, May 1974.

[LAMP78] Lamport, L., "Time, Clocks, and the ordering of events in a distributed system", Communication of the ACM, 21, 7, 558-564, July, 1978.

[LEIN80] Leinbaugh, D. W., "Guaranteed Response Times in a Hard Real-Time Environment," IEEE Transactions on Software Engineering, Vol. SE-6, Jan 1980.

[LEIN82] Leinbaugh, D.W. and Yamini, M., "Guaranteed Response Times in a Distributed Hard-Real-Time Envoronment", Proc. of the Real-Time Systems Symposium, Dec 1982.

[LIU73] Liu, C. L., amd J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," JACM, Vol. 20, No. 1, Jan 1973.

[MOK78] Mok, A.K. and Dertouzos, "Multiprocessor Scheduling in a Hard Real-Time Environment", Proc. of the Seventh Texas Conference on Computing Systems, Nov 1978.

[MUNT70] Muntz, R. R., and E. G. Coffman, "Preemptive Scheduling of Real-Time Tasks on Multiprocesssor Systems," JACM, Vol. 17, No. 2, April 1970.

[SMIT80] Smith, R.G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", IEEE Transactions on Computers C-29, 12, Dec 1980.

[TEIX78] Teixeira, T., "Static Priority Interrupt Scheduling," Proc. of the Seventh Texas Conference on Computing Systems, Nov 1978.

8.0 Distribution List

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2 copies
Commander USAERADCOM ATTN: DELSD-L-S Fort Monmouth, NJ 07703	2 copies
Commander USACECOM ATTN: DRSEL-COM-RF (Dr. Klein) Fort Monmouth, NJ 07703	1 copy
Commander USACECOM ATTN: DRSEL-COM-D (E. Famolari) Fort Monmouth, NJ 07703	1 copy
Commander USACECOM ATTN: DRSEL-COM-RF-2 (C. Graff) Fort Monmouth, NJ 07703	10 copies

FILM

3-8